Post-silicon failing-test generation through evolutionary computation

Post-Silicon Failing-Test Generation through Evolutionary Computation

Ernesto Sanchez, Giovanni Squillero. Alberto Tonda
Politecnico di Torino
Torino, ITALY
{ ernesto.sanchez, giovanni.squillero, alberto.tonda } @polito.it

***Abstract*— The incessant progress in manufacturing technology is posing new challenges to microprocessor designers. Several activities that were originally supposed to be part of the pre-silicon design phase are migrating after tape-out, when the first silicon prototypes are available. The paper describes a post-silicon methodology for devising functional failing tests. Therefore, suited to be exploited by microprocessor producer to detect, analyze and debug speed paths during verification, speed-stepping, or other critical activities. The proposed methodology is based on an evolutionary algorithm and exploits a versatile toolkit named µGP. The paper describes how to take into account complex hardware characteristics and architectural details of such complex devices. The experimental evaluation clearly demonstrates the potential of this line of research.**

I. INTRODUCTION

Nowadays, manufacturing technology is advancing at a faster pace than designing capability, posing unprecedented challenges in the arena of integrated circuits. Due for example to limitations in current timing analysis tools, the comprehensive analysis of a complex chip can only be performed after tape-out. In industrial practice, once manufacturing is completed and first silicon is produced, the early chips are sent back to their design teams. Like the well-known *post-silicon verification*, several activities that were originally supposed to be part of the pre-silicon design phase are nowadays migrating to the post-silicon time. The cost of manufacturing prototypical devices is indeed enormous, but this practice is not an option. Designers acknowledged that "very few chips ever designed function or meet their performance goal the first time" [1].

Non-deterministic effects, such as manufacturing variability, are posing great challenges to the designers. It has been long known that several physical defects only appear when the device operates at full speed [2], but nowadays also design criticalities become apparent only at high frequencies. Even worse, they appear only occasionally and possibly only in a percentage of the manufactured chips. "Finding the root cause of at-speed failures remains one of the biggest challenges in any high-performance design", stated Rob Aitken in his *editor's note* for [3].

Microprocessors are a paradigmatic example of the current scenario: devices for the desktop market contain billions of transistors, implement quite complex microarchitectures, and operate into the microwave frequency range.

A *speed path* is a path that limits the performance of a chip because a faster clock would cause an incorrect behavior with a relevant probability. Speed paths may be the location where potential design fixes should be applied, and may indicate places where potential holes in the design methodologies or manufacturing technologies exist. At design time, the slowest logic path in a circuit is known as the *critical path*, and it can be quite easily determined. However, in complex multicore/multithreaded designs it has been recognized that critical paths reported from the pre-silicon timing analysis tools poorly correlate with the actual speed paths. The reason is that obtaining accurate models for nanometer processes is difficult, if not nearly impossible. Analysis algorithms are also approximated and oversimplified because of the complexity involved. Finally, timing behavior on the silicon is a result of several factors mingled together, and during the pre-silicon analysis it is not feasible to consider multiple factors simultaneously [4] [5] [6].

To meet today's performance requirements, the design flow of a modern microprocessor goes through several iterations of frequency pushes prior to final volume production. Such a process is called *speed stepping*. The identification and the debug of speed paths is an essential part of speed stepping. A *failing test* is defined as a sequence of operations that uncovers an incorrect behavior when run at high frequency. Failing tests may be, for example, sequences of inputs to be applied to the microprocessor pins by an automatic test equipment (ATE). In industrial practice, such tests are crafted by engineers starting from the pre-silicon verification test suite; generated by pre-silicon specialized tools, or automatic test pattern generators (ATPGs); or created *on silicon*, tackling the actual devices [7] [8].

Interestingly, the instruction sets of microprocessors have been successfully exploited to tackle *path-delay faults*. The underlying idea of these works is that executing a set of carefully designed programs may effectively uncover timing issues. The strengths of the methodology are that the execution of such test programs is per se *at-speed* and requires no additional hardware, or complex and expensive ATEs. No attempts, however, have been reported to devise failing tests directly at the instruction level. No one has yet proposed a post-silicon methodology able to automatically generate a test program that stresses a speed path causing a detectable functional failure.

A *software-based speed-path failing test* is defined as an assembly-language program that produces the correct result

only while the microprocessor operating frequency is below a certain threshold. As soon as the frequency is pushed above the threshold, the result yielded by the program becomes incorrect. Let us denote the threshold for a given program as its *functional frequency threshold*, because the incorrect behavior is functionally observable. That is, it can be theoretically detected by observing the values stored in the main memory and registers. Clearly, the diagnostic capability of a software-based speed-path failing test increases as its functional frequency threshold decreases. A test that produces a failure at a relatively low frequency is preferable to a test that fails only at very high frequencies.

This paper shows for the first time a fully automatic methodology for devising functional speed-path failing tests. It demonstrates that highly effective software-based tests could be generated directly on-silicon exploiting an evolutionary algorithm. The result advocates for the exploitation of the methodology inside the manufacturer's facility during the speed stepping phase.

Sections 2 and 3 describe the proposed methodology, detailing the adopted evolutionary algorithm. Sections 4 illustrates the feasibility study and report the obtained results. Section 5 concludes the paper, sketching the future directions of the research.

II. Generation and Evaluation of Test Programs

The proposed approach for generating software-based speed-path failing tests is *feedback-based*. Candidate test programs are created without a rigid scheme, and evaluated on the target microprocessor. The data gathered are fed back to the generator and used to generate a new, enhanced set of candidate solutions. The process is then iterated.

To exploit a feedback-based mechanism it is required to evaluate the goodness of each candidate test. As stated before, a software-based speed-path failing test is as good as it fails at low frequencies, and the key parameter in evaluating a test is its functional frequency threshold. However, it should not be forgotten that variability vexes verification engineers. A failing test may not fail always at the same frequency, even if all controllable parameters are exactly reproduced. The variability of speed paths may be caused by non-deterministic factors, such as noise, die temperature or small fluctuation in the external power. Some design criticalities may appear only under particularly unfavorable conditions. All experiments need to be repeated at least several times, when not on different devices.

Consequently, besides the lowest functional frequency threshold detected amongst the repeated experiments, an additional parameter in evaluating a test is the percentage of runs that actually failed at that frequency. It is intuitively plausible that a test failing half of the times at a certain frequency is more useful that a test that fails only once every thousands experiments.

For the sake of a feasibility study, however, the evaluation of candidate tests was performed decreasing the core voltage, a practice known as *undervolting*. Reducing the voltage increases, roughly speaking, the time required to switch between logic values [9]. It must be noted that increasing frequency and reducing voltage involves significantly different phenomena in the physical world, especially where not all paths have the same Vcc sensitivity or where paths are interconnect dominated. However, there is no *conceptual difference* between overclocking and undervolting from the point of view of the proposed algorithm.

III. Evolutionary Core

The optimizer used in the feed-back approach is an *evolutionary algorithm*, that is, some of its internal mechanism loosely mimic principles of the Neo-Darwinian paradigm, namely variation, inheritance, and selection. The toolkit exploited in this work is called μGP (MicroGP) [10], available under the *GNU Public License* from *Sourceforge*[1]. The toolkit has already been used in several works[2], its description is out of the scope of this text.

The efficacy of a evolutionary algorithms depend on several factors, but the two most important ones are: what feedback is used to evaluate candidate solutions, termed *fitness* by the evolutionary algorithm scholars; what is encoded inside individuals. While exploiting an evolutionary approach is *per-se* of little interest, selecting and tuning such elements can effectively enable to find a solution.

A. Fitness Function

The fitness function must not only be able to evaluate candidate solutions, but also be able to rank them, identifying the more promising ones. In these kind of algorithms, candidate solutions are optimized *through the accumulation of slight but useful variations* [11]. Thus, individuals must be distinguishable because of different fitness for evolution to take place.

In the proposed framework, the first and most important component of the fitness is simply the functional voltage threshold. The second is the number of failures detected over the *R* repetitions at the maximum voltage.

Similarly to *software-based self test*, candidate test programs include a mechanism that helps checking their own correctness: all the results of the calculations performed by the test program are compacted in a single signature using a hash function. The evaluator first runs the test program in safe conditions, i.e., at full power, and store the signature. Then it runs the program again at decreasing CPU core voltages, checking that the signature is not modified. As soon as a difference is detected, the functional voltage threshold is recorded. The whole process is repeated *R* times to tackle variability. It must be stressed out that the actual result of the calculations is of no interest, the only relevant detail being that it changes when the test is executed undervolting the CPU below the functional core voltage.

Operatively, μGP creates assembly functions that are assembled and linked with a *manager* module. These functions contain a loop that execute *L* times a set of instructions. The instructions themselves are devised by the evolutionary core, while the framework is fixed. At the end of the loop, before the

[1] http://sourceforge.net/projects/ugp3/

[2] See http://www.cad.polito.it/pap/keyword/MicroGP.html for an updated list of references

next iteration, the values in the registers are used to update the signature.

B. Internal Representation

The internal representation is another key aspect. The evolutionary algorithm must be given the opportunity to generate useful solutions. Modern processors may implement a multithreaded design; or they can exploit a multicore architecture; or even both. A single individual is composed of different independent functions.

Inside each thread, the assembly instructions made available to µGP can be divided in three main classes: *integer instructions*; legacy *x87 instructions*; *single-instruction/multiple-data* (SIMD) *instructions*. Not surprisingly, SIMD instructions are particularly critical during speed stepping: the complex calculations involved by these instructions cause data to go through several functional units, and the resulting *datapaths* are prone to be source of problems when the operating frequency is increased.

Cache memories must be also taken into account as well, since there may be a significant difference in performance and power consumption between a L1 cache hit and a L1 cache miss. The µGP was given the possibility to generate cache hits and cache misses through a set of variables carefully spaced in memory. It must be noted that the goal of adding such variables is to let the evolutionary core to control the cache activity, but no suggestions are given on how to exploit them. µGP would find autonomously which sequence of operations is more useful to generate a failing test.

IV. Experimental Evaluation

While no working methodology for functional failing-test generation has been reported in the specialized literature yet, a related problem is faced by a community of computer enthusiasts. *Overclockers* try to push the performance by increasing the operating frequencies of their microprocessors and the CPU core voltages [12]. However, after pushing their computers to astonishing frequencies, they need to assess the stability of their systems. The test suites that are used to stress the systems and highlight criticalities may be regarded as generic fail tests not focused on a specific microprocessor. Thus, they can be used as a baseline to evaluate the performances of the proposed methodology.

Most of the information about stability stress tests is available through forums and web sites on the internet, with few or none official sources. However, there is quite a generalized agreement in the overclockers community on these tools. *SuperPI* and *CPU BurnIn* are two applications that exploited mathematical calculations, they use no SIMD instructions and are single threaded. These two programs are rather old, but have been included for the sake of comparison. *Prime95* was originally created to find *Mersenne prime numbers*, and ver the years, it has become extremely popular among overclockers as a stability test. *LinX*, *IntelBurnTest*, and *OCCT* are all based on an Intel benchmark distributed with the *Math Kernel Library*.

While all the stability tests are quite different, a common point is that modern ones do extensive SIMD calculation. Another common point is their ability to increase the temperature of the microprocessor. It is well known that high temperature may cause design criticalities to become manifest and the circuit to operate incorrectly. However, while such an effect is sensible when assessing the stability of a system, it may not be desirable when the goal is to find a failing test during speed stepping. The main reason is that the failing test should be as repeatable as possible, while increasing the temperature also increase non-deterministic phenomena.

TABLE I. Failing-test required time for E2180

CORE V	Prime95	IntelBurnTest	LinX	OCCT	µGP
1.2625	1”	2’	2’	3”	≤ 1”
1.2750	6”	2’	2’	4”	2”
1.2875	4’	4’	2’	7’	2”
1.3000	> 10’	7’	7’	> 10’	10”
1.3125	> 10’	> 10’	> 10’	> 10’	8’
1.3250	> 10’	> 10’	> 10’	> 10’	> 10’

Experiments were run on two different systems. In both cases, the only non-standard devices were in-house manufactured water cooling systems. The first benchmark is an *Intel Pentium Core 2 Duo E2180*, a dual-core microprocessor based on the *Core* architecture. The second benchmark is an *Intel Pentium Core i7-950*, based on *Nehalem* architecture, the successor of the *Core* architecture. It is a quad-core microprocessor, able to run up to 8 threads with simultaneous multithreading.

The failing test devised by the proposed approach on the target system was compared with the state-of-the-art stress tools used by the overclocking community. Results are reported in Table I and Table II. Columns are labeled with the name of the program used to test the system. The last column reports data of the test generated by µGP. Rows indicate the CPU core voltage at which the experiments were run. Cells shows the time required for the given stress test to report a failure. To reduce overheating effects, all tests were stopped after 10 minutes. Thus “more than 10 minutes” means that no failure has been detected. Almost instantaneous failures are reported as “less than 1 second”. All experiments have been repeated 10 times. µGP parameters were the default ones: a population of 30 tests, with about 30 new programs generated in each step.

Table I and table II on the other side, report the comparison against newer stress tests. For the Intel Pentium Core 2 Duo E2180, all programs use two threads, that is, one for each core. For the Intel Pentium Core i7-950, all programs use eight threads, that is, two for each core.

Failing tests devised with the proposed methodology clearly outperform all the other approaches. Remarkably, µGP was asked to find a very fast failing test for a specific microprocessor, and therefore there is no guarantee that the devised program would fail on a different model. Moreover, the test was required to be very short, to avoid heating effects. The temperature of the microprocessor during the experiments never exceeded 40°C for the E2180 and 50°C for the i7-950, while running LINPACK-based stress tests are significantly higher, even with the liquid cooling. On the contrary, stress tests intentionally exploit overheating and are designed to work with different architectures.

The failing test for the E2180 is 614 line long. The two functions executed by the two cores are respectively 280 and 235 line long. The remaining lines are mainly used to define and initialize variables or other program parts. The failing test for the i7-950 is 997 line long. The four functions executed by the four cores are respectively 236, 206, 153, 174 line long. The remaining lines are mainly used to define and initialize variables or other program parts.

TABLE II. FAILING-TEST REQUIRED TIME FOR I7-950

CORE V	Prime95	IntelBurnTest	LinX	OCCT	µGP
1.21250	6’	1’	2’	4’	<1’’
1.21875	>10’	>10’	4’	5’	<1’’
1.22500	>10’	>10’	>10’	>10’	<1’’
1.23125	>10’	>10’	>10’	>10’	<1’’
1.23750	>10’	>10’	>10’	>10’	<1’’
1.24375	>10’	>10’	>10’	>10’	<1’’
1.25000	>10’	>10’	>10’	>10’	<1’’
1.25625	>10’	>10’	>10’	>10’	1’’
1.26250	>10’	>10’	>10’	>10’	1’’
1.26875	>10’	>10’	>10’	>10’	3’’
1.27500	>10’	>10’	>10’	>10’	3’’
1.28125	>10’	>10’	>10’	>10’	3’’
1.28750	>10’	>10’	>10’	>10’	5’’
1.29375	>10’	>10’	>10’	>10’	30’’
1.30000	>10’	>10’	>10’	>10’	2’
1.30625	>10’	>10’	>10’	>10’	5’
1.31875	>10’	>10’	>10’	>10’	>10’
1.32500	>10’	>10’	>10’	>10’	>10’

µGP required about 5 hours to generate the failing test for the E2180, and 40 hours for the i7-950. The difference in time can be explained taking into account the greater number available steps, and the length of the test itself.

A. Feedback from the Overclockers Community

The generated tests were made available to the overclockers community as *ultra-fast stability test* [3] . Although not systematic, the feedback fully confirmed our claims: results on i7-950 microprocessors show the superiority of the µGP test. Similar results are achieved on i7-920 units. Interestingly, the failing test is not effective on the i7-860 family. Thus, it is reasonable to presume that the test stress some microarchitectural features present only in certain family.

V. CONCLUSIONS AND FUTURE WORKS

The paper proposed an efficient post-silicon methodology for devising functional failing tests. Experimental results clearly demonstrate that tests are able to highlight criticalities specific of the target microarchitecture: they work equally well with all i7-950 units, but not with devices of different families. More interestingly, it is able to do it without any information about the design. Such result is not completely surprising: µGP already managed to stress specific microarchitectural features tackling a Pentium 4 as a mere black box [13].

The proposed methodology could be exploited by microprocessor manufacturers during verification or speed stepping. The knowledge of the internal design and the physical access to the device under test would be necessary to continue on this line of research.

On the other hand, the methodology could also be used to generate a fast test able to check the reliability of a system. This usage, can be important for the incoming inspection of a set of purchased devices.

Future works include enhancing the evolutionary algorithm, letting it set the number of repetitions in each test *L*. The interaction between x87 and SIMD instructions also deserves a closer examination. A customized version of the µGP requiring no operating systems can be devised in order to more easily run experiments on the microprocessor. Also, the signature could be improved by including more information on the state of the execution, such as the internal performance monitor.

[3] http://www.cad.polito.it/research/Evolutionary_Computation/Overclocking.html

References

[1] R. McLaughlin, S. Venkataraman, and C. Lim, "Automated Debug of Speed Path Failures Using Functional Tests," in *The 27th IEEE VLSI Test Symposium*, 2009, pp. 91-96.

[2] K. M. Thompson, "Intel and the myths of test," *IEEE Design & Test of Computers*, vol. I, no. 13, pp. 79-81, 1996.

[3] K. Killpack, S. Natarajan, A. Krishnamachary, and P. Bastani, "Case Study on Speed Failure Causes in a Microprocessor," *IEEE Design & Test of Computers*, vol. 25 , no. 3, pp. 224-230, 2008.

[4] J. Zeng, M. Abadir, J. Bhadra, and J. Abraham, "Full chip false timing-path identification," in *2000 IEEE Workshop on Signal Processing Systems*, 2000, pp. 703-711.

[5] K. Killpack, C. Kashyap, and E. Chiprout, "Silicon Speedpath Measurement and Feedback into EDA flows," in *44th Design Automation Conference*, 2007, pp. 390-395.

[6] N. Callegari, L. .-C. Wang, and P. Bastani, "Speedpath analysis based on hypothesis pruning and ranking," in *46th ACM/IEEE Design Automation Conference*, 2009, pp. 346-351.

[7] L. Lee, L.-C. Wang, P. Parvathala, and T. M. Mak, "On Silicon-Based Speed Path Identification," in *Proceedings of the 23rd IEEE VLSI Test Symposium*, 2005, pp. 35-41.

[8] J. Zeng, J. Wang, C.-Y. Chen, M. Mateja, and L. .-C. Wang, "On evaluating speed path detection of structural tests," in *11th International Symposium on Quality Electronic Design (ISQED)*, 2010, pp. 570-576.

[9] A. B. Bhattacharyya, *Compact MOSFET Models for VLSI Design*. Wiley, John & Sons, 2009.

[10] E. Sanchez, M. Schillaci, and G. Squillero, *Evolutionary Optimization: The µGP Toolkit*. Springer, 2011, ISBN: 978-0-387-09425-0.

[11] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: Murray, 1859.

[12] B. Colwell, "The Zen of overclocking," *Computer*, vol. 37, no. 3, pp. 9-12, 2004.

[13] W. Lindsay, E. Sanchez, M. S. Reorda, and G. Squillero, "Automatic test programs generation driven by internal performance counters," in *5th International Workshop on Microprocessor Test and Verification*, 2004, pp. 8-13.